This is the accepted version of the paper. The final version will appear in the proceedings of LANGETI'2020

# Testing Practices for Infrastructure as Code

Mohammed Mehedi Hasan
Independent University
Dhaka, Bangladesh
mehedi.bueteee.23@gmail.com

Farzana Ahamed Bhuiyan
Tennessee Tech University
Cookeville, TN, USA
fbhuiyan42@students.tntech.edu

Akond Rahman
Tennessee Tech University
Cookeville, TN, USA
arahman@tntech.edu

## ABSTRACT

Infrastructure as code (IaC) helps practitioners to rapidly deploy software services to end-users. Despite reported benefits, IaC scripts are susceptible to defects. Defects in IaC scripts can cause serious consequences, for example, creating large-scale outages similar to the Amazon Web Services (AWS) incident in 2017. The prevalence of defects in IaC scripts necessitates practitioners to implement IaC testing and be aware of IaC testing practices. A synthesis of IaC testing practices can enable practitioners in early mitigation of IaC defects and also help researchers to identify potential research avenues. *The goal of this paper is to help practitioners improve the quality of infrastructure as code (IaC) scripts by identifying a set of testing practices for IaC scripts.* We apply open coding on 50 Internet artifacts, such as blog posts to derive IaC testing practices. We identify six testing practices that include behavior-focused test coverage, the practice of measuring coverage of IaC test cases in terms of expected behavior. We conclude our paper by discussing how practitioners and researchers can leverage our derived list of testing practices for IaC.

## CCS CONCEPTS

• **Software and its engineering → Software defect analysis**.

## KEYWORDS

configuration as code, devops, empirical study, infrastructure as code, practices, qualitative analysis, testing

**ACM Reference Format:**

Mohammed Mehedi Hasan, Farzana Ahamed Bhuiyan, and Akond Rahman. 2020. Testing Practices for Infrastructure as Code. In *Proceedings of the 1st ACM SIGSOFT International Workshop on Languages and Tools for Next Generation Testing (LANGETI '20), November 8–9, 2020, Virtual, USA*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3416504.3424334

## 1 INTRODUCTION

Continuous deployment (CD) is the process of rapidly deploying software or services automatically to end-users [33]. In CD, if all test cases and quality checks pass, then submitted software and service changes deploy automatically to production servers [33]. One practice that is integral to CD is the practice of infrastructure as code (IaC). IaC scripts are essential to implement an automated deployment pipeline, which facilitates CD [20].

IaC is the practice of automatically defining and managing system configurations through source code [20]. Practitioners use IaC tools, such as Ansible [1] and Chef [2] that provide utilities to implement the practice of IaC [16]. IaC tools provide programming syntax and libraries so that practitioners can specify configuration and dependency information as scripts. As a practice, IaC is emerging and growing in popularity amongst practitioners [28]. As shown in Figure 1, based on Google trends data related to the search term 'infrastructure as code' interest in IaC has steadily increased since 2012.

Use of IaC has resulted in benefits for information technology (IT) organizations, for example, the Enterprise Strategy Group surveyed practitioners in 2016 and reported the use of IaC scripts to help IT organizations on average gain 210% in time savings [3]. As another example, the use of IaC scripts helped the National Aeronautics and Space Administration (NASA) to reduce its multi-day patching process to 45 minutes [9].

Despite reported benefits, IaC scripts are susceptible to defects, which can cause serious consequences, e.g., a defect in an IaC script resulted in an outage worth of 150 million USD for Amazon Web Services (AWS) in 2017 [4]. As another example, execution of a defective IaC script erased home directories of ~270 users in cloud instances maintained by Wikimedia Commons [5].

The prevalence of IaC defects [26] necessitates the testing of IaC scripts. However, not knowing what practices to implement can deter practitioners from adopting IaC testing [16]. Derivation of IaC testing practices can enable practitioners to test IaC scripts effectively and also identify future research avenues that could be of interest to the software engineering research community.

One strategy to identify IaC testing practices is to systematically analyze Internet artifacts, such as blog posts and video presentations that discuss IaC testing. Practitioners often report what practices they use in Internet artifacts instead of academic forums, such as research conferences [11, 13]. In prior work, researchers have acknowledged the value of Internet artifacts in deriving practices and analyzed Internet artifacts to summarize security practices used in DevOps [37, 41] and practices used for continuous deployment [33]. Analysis of Internet artifacts can be useful to identify IaC testing practices, a research topic that remains under-explored [28]. Our hypothesis is that by analyzing Internet artifacts we can identify a list of testing practices for IaC.


*LANGETI '20, November 8–9, 2020, Virtual, USA*

ACM ISBN 978-1-4503-8123-9/20/11…$15.00
https://doi.org/10.1145/3416504.3424334

[1] https://www.ansible.com/
[2] https://chef.io/
[3] https://puppet.com/resources/analyst-report/the-economic-benefits-puppet-enterprise
[4] https://www.npr.org/sections/thetwo-way/2017/03/03/518322734/

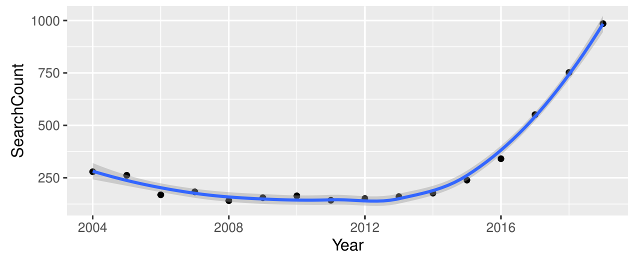


**Figure 1: Growing interest in IaC based on Google trends data. The x and y axis respectively, presents the year and the search count for each year related to the term 'infrastructure as code'.**

*The goal of this paper is to help practitioners improve the quality of infrastructure as code (IaC) scripts by identifying a set of testing practices for IaC scripts.*

We answer the following research question: *What testing practices can be used for infrastructure as code scripts according to practitioners?* We apply open coding [34] on 50 Internet artifacts to derive IaC testing practices.

**Our contribution** is a list of IaC testing practices.

We organize the rest of the paper as follows: Section 2 provides IaC background and prior research related to IaC. We provide our empirical study in Section 3. Section 4 provides a discussion of how practitioners and researchers can leverage our findings. Finally, we conclude in Section 5.

## 2 BACKGROUND AND RELATED WORK

We provide background and discuss related work in this section.

### 2.1 Background

IaC is the practice of automatically defining and managing deployment environments, system configurations, and infrastructure through source code [20]. Before IaC tools were available, system operators used to create custom configuration scripts, which were not developed and maintained using a systematic software development process [40]. The 'as code' suffix refers to applying development activities considered to be good practices in software development, such as keeping scripts in version control, testing, and submitting code changes in small units [25]. With the availability of cloud computing resources such as AWS [5], the development and maintenance of deployment scripts became complex, which motivated IT organizations to treat their configuration scripts as regular software source code. IaC scripts are also referred to as configuration scripts [38] or configuration as code scripts [30]. With respect to maintainability, debugging, and testing, IaC is different to that of general purpose programming languages (GPLs) [16, 21].

Multiple tools, such as Ansible and Chef exist to implement the practice of IaC. A 2019 survey with 786 practitioners reported Ansible as the most popular language to implement IaC, followed by Chef [6] [7]. Both, Ansible and Chef provide multiple libraries to manage infrastructure and system configurations. In the case of Ansible, developers can manage configurations using 'playbooks', which uses YAML files to manage configurations. For example, as shown in Figure 2, an empty file '/tmp/sample.txt' is created using the 'file' module provided by Ansible. The properties of the file such as, path, owner, and group can also be specified. The 'state' property provides options to create an empty file using the 'touch' value.

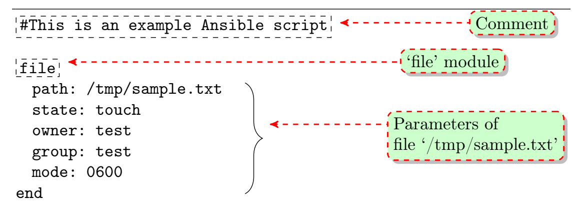


**Figure 2: Annotation of an example Ansible script.**

In the case of Chef, configurations are specified using 'recipes', which are domain-specific Ruby scripts. Dedicated libraries are also available to maintain certain configurations. As shown in Figure 3, using the 'file' resource, an empty file '/tmp/sample.txt' is created. The 'content' property is used to specify the content of the file is empty.

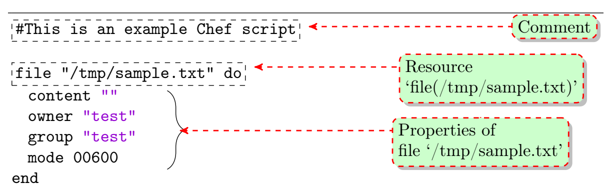


**Figure 3: Annotation of an example Chef script.**

### 2.2 Related Work

Our paper is closely related to prior research on IaC scripts. Sharma et al. [38], Schwarz [36], and Bent et al. [42], in separate studies investigated code maintainability aspects of Chef and Puppet scripts. Hanappi et al. [17] investigated how the convergence of IaC scripts can be automatically detected, and proposed an automated model-based detection framework for convergence. Rahman et al. [26] constructed a defect taxonomy for IaC scripts that included eight defect categories. In another work, Rahman et al. [27] identified five development anti-patterns for IaC scripts. Guerriero et al. [16] identified lack of testing practices as a barrier for testing IaC scripts. Ikeshita et al [23] reported testing of IaC scripts can be time-consuming, and proposed a test suite reduction technique for IaC scripts. Hummer et al. [21] observed that testing in IaC is different to that of GPLs, as testing in IaC necessitates testing of production environments. In another work, Hummer et al. [22] created a tool to automatically test the idempotency of IaC scripts.

[5] https://aws.amazon.com/

[6] https://info.flexerasoftware.com/SLO-WP-State-of-the-Cloud-2019

[7] https://www.techrepublic.com/article/ansible-overtakes-chef-and-puppet-as-the-top-cloud-configuration-management-tool/

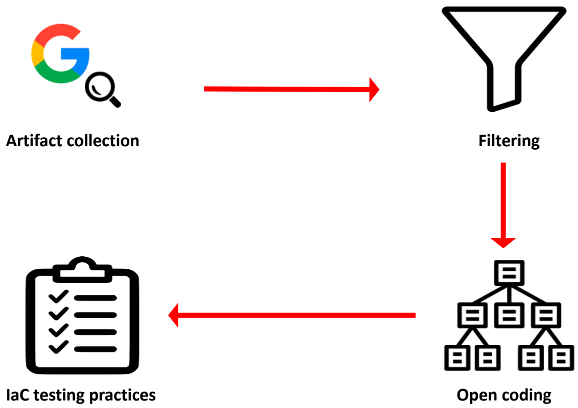


**Figure 4: Methodology to identify IaC testing practices.**

Rahman and Williams [1] and Palma et al. [8] in separate restudies identified code metrics that show correlation with defective IaC scripts. In separate papers, Rahman et al. identified insecure coding patterns for Puppet scripts [29], and Ansible and Chef scripts [31]. Rahman and Williams [32] characterized defective IaC scripts using text mining and created prediction models using text feature metrics. Rahman et al. [28] conducted a systematic mapping study with 32 IaC-related publications and identified limited number of testing tools for IaC scripts. They [28] also observed a lack of research related to best practices in the domain of IaC.

From the above-mentioned discussion, we observe a lack of research related to IaC testing practices, which we have addressed in our paper. Our paper complements the testing-related observations and recommendations related to IaC reported by Rahman et al. [28].

## 3 EMPIRICAL STUDY

In this section we provide the methodology and results for our research question: *What testing practices can be used for infrastructure as code scripts?*

### 3.1 Methodology

We conduct our empirical study by systematically analyzing Internet artifacts, such as blog posts and videos [19]. Systematic investigation of these artifacts has been previously used by researchers to identify practices used in continuous deployment [33]. Previously, researchers [19, 33] have observed that practitioners tend to report their experience and their perception of best practices in blog posts. Our hypothesis is that by analyzing artifacts we will be able to synthesize IaC testing practices.

As summarized in Figure 4, we conduct our empirical study using three steps described below:

**Step#1-Internet Artifact Collection**: First, we use the Google search engine to collect Internet artifacts. We use the following search strings:

- "testing infrastructure as code"
- "testing ansible playbooks"
- "testing ansible roles"
- "testing chef cookbook"
- "testing terraform code"
- "testing best practices for ansible scripts"
- "testing best practices for chef scripts"
- "testing best practices for puppet scripts"
- "testing best practices for terraform scripts"

Our first search string is 'testing infrastructure as code'. Using our first search string, we collect the first 50 search results sorted by relevance according to Google search engine. From the collected first 50 search results we observe Internet artifacts to mention testing for 4 languages: Ansible, Chef, Puppet, and Terraform. To include artifacts that discuss testing practices for Ansible, Chef, Puppet, and Terraform we added the other 8 search strings. The above-mentioned approach is similar to forward snowballing [43] where we start with a set of search strings and generate more search strings until no new search string is found. After generating the above-mentioned 8 search strings we are unable to generate new search strings. We use these search strings in the next step to collect necessary Internet artifacts. The search string derivation process is conducted by the first author.

We collect the first 100 search results for each search string sorted by relevance according to Google search engine. We collect the first 100 search results because from manual inspection we identify irrelevant search results to appear after the first 100 search results. The first author manually inspect the top 250 search results for each search string to confirm that the top 100 search results would suffice to identify relevant IaC testing practices. Altogether, we collect 900 search results from the 9 search strings. All the search results are collected using the browser's incognito browsing mode to mitigate search result bias.

**Step#2-Internet Artifact Filtering**: We systematically apply filtering to identify Internet artifacts related to IaC testing. *First*, we remove duplicate Internet artifacts collected from our search results. *Second*, we check if the Internet artifact is available for reading. Artifacts can be inaccessible, and we exclude such search results. *Third*, we manually read the content of the collected search to determine if the Internet artifact discusses about IaC testing. We exclude Internet artifacts that do not explicitly discuss IaC testing. The artifact filtering process is conducted by the first author.

**Step#3-Open Coding**: We apply open coding [6] on the collected Internet artifacts, which we obtain from Step#2. In open coding a rater observes and synthesizes patterns within unstructured text [6]. The first author, who has six years of professional experience in software engineering conducts open coding. During open coding the first author read the content of each artifact to generate sub-categories. Later, the sub-categories are merged based on similarities to derive categories. We apply open coding because using open coding the rater can determine (i) if the collected Internet artifacts are in fact actually related to IaC testing practices, and (ii) identify testing practices used for IaC. In our analysis, one Internet artifact can include multiple practices.

*Rater verification*: The process of deriving categories is susceptible to bias. We mitigate this bias by allocating two more raters rater: the second and last author of the paper. Both rater separately applies closed coding [6] on the collected 50 Internet artifacts. The second author is a third year PhD student with a professional experience of 2 years in software engineering. The last author has 7 years of experience in IaC, and a professional experience of 5 years in software engineering. For each of the 50 artifacts, both raters individually examine if the artifact includes a discussion related to the categories identified by the first author. We calculate the agreement between the first and the second author, and the first and the last author using Cohen's Kappa [4].

For the 50 Internet artifacts the Cohen's Kappa is 0.68 between the first and second author, which suggests 'moderate' agreement, according to Landis and Koch [24]. Reasons for disagreements are attributed to the second author's lack of familiarity with the topic. The agreement rate between the first and last author is 1.0.

## 3.2 Results

For each of the 9 search strings we collect the 100 most relevant search results on December 2019. From our set of 900 search results we remove duplicates and obtain 228 Internet artifacts. Next, we check for availability and find 223 artifacts to be available. Next, we read each of the 223 artifacts and identify 50 artifacts to actually discuss IaC testing practices. A breakdown of the Internet artifact categories is available in Table 1. The constructed dataset is available online [2].

**Table 1: Distribution of Internet Artifacts**

| Type of Artifact | Count |
|---|---|
| Blog | 36 |
| Stack Overflow | 4 |
| Slideshare | 4 |
| Github Repository | 3 |
| Video | 3 |

**Testing Practices for IaC**: We identify six practices for IaC testing. The count of artifacts that mention each practice is provided in Table 2. The temporal evolution of the six practices is provided in Table 3. We describe each practice below:

**Table 2: Count of artifacts mentioning practices**

| Practice name | Count of artifacts |
|---|---|
| Use of Automation | 33 |
| Sandbox Testing | 23 |
| Testing Every IaC Change | 22 |
| Behavior-focused Test Coverage | 16 |
| Avoiding Coding Anti-patterns | 15 |
| Remote Testing | 3 |

I. ***Avoiding Coding Anti-patterns***: The practice of avoiding coding anti-patterns while developing testing code so that test code for IaC is easier to maintain and technical debt is reduced. Example coding anti-patterns include long statements and missing default block in switch statement. Identification of coding anti-patterns in IaC test scripts can be performed using linters. Example linters include 'ansible-lint'[39], 'Foodcritic' [7], and 'tflint'[15] respectively, for Ansible, Chef, and Terraform.

II. ***Behavior-focused Test Coverage***: The practice of measuring coverage of IaC test cases in terms of expected behavior. For coverage measurement of IaC testing, practitioners suggest the use of behavior i.e., what is the expected output of IaC scripts, and generate test cases accordingly so that the expected output is satisfied by the script of interest. The quality of the test cases are evaluated based on if the IaC script of interest follows expected behavior.

While discussing behavior-driven testing for IaC scripts, one practitioner [35] argued for 'robust expectations' stating, "*The art here is to make the expectations robust enough to survive irrelevant changes in the system, while is still sensitive enough to detect actual problems with the code*". To facilitate behavior-driven testing for IaC, practitioners have mentioned tools, such as TestInfra and InSpec. With TestInfra practitioners can write test code in python to verify the states of infrastructure using pytest utilities[18]. InSpec uses Ruby and provide Ruby-based plugins to verify infrastructure states.

III. ***Remote Testing***: The practice of testing IaC scripts in remote environments, for example testing IaC scripts in an AWS instance. Practitioners stated that testing IaC scripts only in local environments can be limiting because an IaC script may execute correctly in a local environment, but erroneously in a remote environment. One practitioner [35] emphasized on remote testing by stating, "*By running the tests on real systems, you can determine whether your application responded correctly in a realistic configuration.*"

Practitioners [3] have suggested the use of testing tools, such as Molecule [8] to perform remote testing because they perceive Molecule to emulate testing on actual systems, such as AWS.

IV. ***Sandbox Testing***: The practice of testing IaC scripts in isolation so that provisioned production infrastructure does not get impacted. Practitioners have reported that during testing of IaC scripts, provisioned instances might be inadvertently updated or deleted. The practice of sandbox testing helps practitioners to not impact any production instances. As a rule of thumb, practitioners suggest isolating development, staging, and production environments, so that testing activities in one environment does not impact the other. One practitioner [3] stated "*I would actually create a completely separate place to run these tests, where you don't have to worry [about] what happens if I accidentally delete the wrong thing or create the wrong thing?*".

Practitioners have mentioned that tools, such as EC2 driver and molecule-gce provided by Molecule that can be used to set up sandbox testing in public and private cloud environments. Practitioners have also commented on the life cycle of sandbox testing. After testing of IaC scripts, if the sandbox is not required anymore, it should be destroyed to avoid unnecessary pricing. For example, in 2016, Bauer Media's employees inadvertently forgot to delete necessary instances after testing, which resulted in unnecessary costs [9].

V. ***Testing Every IaC Change***: The practice of testing whenever there are changes in IaC scripts. Practitioners suggested application of continuous integration (CI) for IaC scripts so that every change in IaC scripts is validated and integrated. Practitioners have acknowledged that testing every change using CI can be lengthy as 20 minutes, but the benefits outweigh the limitations. Practitioner-reported benefits of testing every change in IaC scripts include (i) obtaining faster feedback on code changes, (ii) early identification of dependency defects, such as container versions, and (iii) able to

[8]https://molecule.readthedocs.io/en/latest/

[9]https://www.itnews.com.au/news/how-bauer-media-dealt-with-public-cloud-bill-shock-420319

**Table 3: Usage of IaC Testing Practices Reported Every Year**

| Practice | 2015 | 2016 | 2017 | 2018 | 2019 |
|---|---|---|---|---|---|
| Avoiding Coding Anti-patterns | 1 | 3 | 4 | 2 | 5 |
| Behavior-focused Test Coverage | 0 | 1 | 3 | 6 | 6 |
| Remote Testing | 0 | 0 | 0 | 3 | 0 |
| Sandbox Testing | 2 | 2 | 6 | 6 | 7 |
| Test Every IaC Change | 1 | 3 | 6 | 4 | 8 |
| Use of Automation | 1 | 3 | 10 | 9 | 10 |

test IaC scripts for multiple platforms. Practitioners suggest that the above-mentioned benefits may also impact the open source community, where volunteers contribute code using pull requests [14].

Even if nothing is changed in IaC scripts, practitioners still suggest testing scripts at regular intervals to examine if environmental changes, such as operating system updates and infrastructure version updates cause problems. One practitioner [12] mentioned using a 'weekly cron schedule' for testing IaC scripts: "*all the common usage scenarios are thoroughly and automatically tested—not only on every pull request and commit, but also on a weekly cron schedule*".

Practitioners can use unit testing tools, such as Molecule for Ansible, and CI tools, such as Travis CI [10] to implement this practice.

VI. ***Use of Automation***: The practice of applying automation to implement testing for IaC scripts with IaC-specific tools, such as Molecule. From our analysis, we observe practitioners to advocate for automation because automation helps in reducing manual efforts. A practitioner [3] emphasized on the use of automation by stating "*I think this is the general law: infrastructure code that does not have automated tests is broken. I don't mean it's going to be broken in the future. I mean it's probably broken right now.*"

From our analysis of Internet artifacts we observe practitioners to mention availability of tools that can help in automated testing of IaC scripts. We observe available IaC testing tools to be language-dependent, for example, Molecule and Test Kitchen [11] [12] are automated testing tools, respectively, for Ansible and Chef.

The above-mentioned automated testing tools also include other features, such as syntax checking and environment setup. For example, Molecule uses ansible-lint for syntax checking and Docker for setting up environment. To setup environment, Test Kitchen uses Virtualbox instead of Docker. Terratest claims to help practitioners in setting up cloud providers and test such cloud setup. In short, available automated testing tools differ from each other with respect to (i) availability of features and (ii) the underlying technology that enables the implementation of such features.

**Limitations**: We discuss the limitations of our paper as following:

- *External Validity*: Our list of practices and the collection of Internet artifacts used to derive such practices are not comprehensive. We may have missed practices that could have been identified by practitioner interviews.
- *Conclusion Validity*: The derivation process of the practices is subject to rater bias, which we mitigate using rater verification. Furthermore, our derived practices is limited to the search process using which we collected the set of 50 Internet artifacts.

[10]https://travis-ci.org/
[11]https://docs.chef.io/workstation/kitchen/
[12]https://github.com/gruntwork-io/terratest

## 4 IMPLICATIONS

We envision future directions by discussing how practitioners and researchers can leverage our research findings.

*Implication for practitioners*: Our categorization of six practices can be useful for:

- Practitioners who want to adopt IaC and are seeking guidance on how to conduct IaC testing;
- Practitioners who are already using IaC but seek guidance on IaC testing practices and necessary tools to implement such practices; and
- Practitioners who want to compare their use of testing practices with what is being used by other practitioners.

*Implication for researchers*: Our paper lays the groundwork to conduct further research described below:

- Following recommendations from Rahman et al. [28], we advocate researchers to collaborate with practitioners for constructing a comprehensive set of testing practices for IaC scripts. Our list is preliminary, which can be extended to construct a comprehensive list of practices.
- Researchers can investigate how many of the identified six practices are being adopted at what frequency in the open source and proprietary domain.
- Researchers can investigate the challenges of adopting IaC testing practices using mixed-method analysis [10], where researchers can conduct online surveys as well as semi-structured interviews. Such analysis can reveal the need for better tools and techniques upon which researchers can focus.
- Future research can investigate if IaC script quality is correlated with the usage of testing practices. While source code metrics [1] and semantics [32] of IaC script quality have been studied, the relationship between IaC script quality and testing remains unknown.
- Using empirical studies researchers can investigate if adoption of identified testing practices is beneficial for automated infrastructure provisioning.
- Researchers can categorize and quantify testing anti-patterns that occur in IaC scripts.

## 5 CONCLUSION

Lack of testing can introduce defects in IaC scripts, which in turn can have serious consequences. A synthesis of IaC testing practices can be helpful for practitioners to mitigate defects in IaC scripts.

We conducted an empirical study with 50 Internet artifacts and identified 6 IaC testing practices. Use of automation tools is the most frequently mentioned practice. While we acknowledge that our list of six practices is not comprehensive, our identified practices showcase emerging results related to IaC testing practices that can be used by practitioners. Furthermore, our findings can be leveraged by the software engineering community to conduct further research in the domain of IaC testing.

## ACKNOWLEDGMENTS

We thank the PASER group at Tennessee Technological University (TTU) for their valuable feedback. This research was partially funded by the National Science Foundation (NSF) award # 2026869 and the Cybersecurity Education, Research and Outreach Center (CEROC) at TTU.

## REFERENCES


[1] Rahman Akond and Williams Laurie. 2019. Source Code Properties of Defective Infrastructure as Code Scripts. *Information and Software Technology* (2019). https://doi.org/10.1016/j.infsof.2019.04.013

[2] Anonymous Authors. 2020. Dataset for IaC Testing Practices Paper. https://figshare.com/s/605c5b636450a29f420e

[3] Yevgeniy Brikman. 2018. 5 Lessons Learned From Writing Over 300,000 Lines of Infrastructure Code. https://www.hashicorp.com/resources/lessons-learned-300000-lines-code/. [Online; accessed 20-Jun-2020].

[4] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[5] Wikimedia Commons. 2017. Incident documentation/20170118-Labs. https://wikitech.wikimedia.org/wiki/Incident_documentation/20170118-Labs. [Online; accessed 27-Jan-2019].

[6] Benjamin F Crabtree and William L Miller. 1992. Doing qualitative research.. In *Annual North American Primary Care Research Group Meeting, 19th, May, 1989, Quebec, PQ, Canada*. Sage Publications, Inc.

[7] Andrew Crump and others of Foodcritic. 2011. About Foodcritic. http://www.foodcritic.io/. [Online; accessed 20-Jun-2020].

[8] Stefano Dalla Palma, Dario Di Nucci, Fabio Palomba, and Damian Andrew Tamburri. 2020. Toward a catalog of software quality metrics for infrastructure code. *Journal of Systems and Software* 170 (2020), 110726. https://doi.org/10.1016/j.jss.2020.110726

[9] Jonathan Davila. 2016. Ansible/NASA Case Study. http://szsb-gl2x.accessdomain.com/fierce/wp-content/uploads/2016/01/NASA-Case-Study-Ansible.pdf. [Online; accessed 20-Jun-2020].

[10] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. 2008. *Selecting Empirical Methods for Software Engineering Research*. Springer London, London, 285–311.

[11] Vahid Garousi and Barış Küçük. 2018. Smells in software test code: A survey of knowledge in industry and academia. *Journal of Systems and Software* 138 (2018), 52–81.

[12] Jeff Geerling. 2018. Testing your Ansible roles with Molecule. https://www.jeffgeerling.com/blog/2018/testing-your-ansible-roles-molecule/. [Online; accessed 20-Jun-2020].

[13] Robert L Glass. 2006. *Software Creativity 2.0*. developer.* Books.

[14] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*. 345–355.

[15] TFLinter Opensource Group. [n.d.]. tflint Github Repo. https://github.com/terraform-linters/tflint. [Online; accessed 20-Jun-2020].

[16] M. Guerriero, M. Garriga, D. A. Tamburri, and F. Palomba. 2019. Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 580–589.

[17] Oliver Hanappi, Waldemar Hummer, and Schahram Dustdar. 2016. Asserting Reliable Convergence for Configuration Management Scripts. *SIGPLAN Not.* 51, 10 (Oct. 2016), 328–343. https://doi.org/10.1145/3022671.2984000

[18] Krekel Holger et al. 2004. pytest: helps you write better programs. https://docs.pytest.org/en/stable/. [Online; accessed 20-Jun-2020].

[19] Sally Hopewell, Mike Clarke, and Sue Mallett. 2005. Grey literature and systematic reviews. *Publication bias in meta-analysis: Prevention, assessment and adjustments* (2005), 49–72.

[20] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation* (1st ed.). Addison-Wesley Professional.

[21] Waldemar Hummer, Florian Rosenberg, Fabio Oliveira, and Tamar Eilam. 2013. Automated testing of chef automation scripts. In *Proceedings Demo & Poster Track of ACM/IFIP/USENIX International Middleware Conference*. 1–2.

[22] Waldemar Hummer, Florian Rosenberg, Fabio Oliveira, and Eilam Tamar. 2013. Testing idempotence for infrastructure as code. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 368–388.

[23] Katsuhiko Ikeshita, Fuyuki Ishikawa, and Shinichi Honiden. 2017. Test suite reduction in idempotence testing of infrastructure as code. In *International Conference on Tests and Proofs*. Springer, 98–115.

[24] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 159–174. http://www.jstor.org/stable/2529310

[25] Kief Morris. 2016. *Infrastructure as code: managing servers in the cloud.* " O'Reilly Media, Inc.".

[26] Akond Rahman, Effat Farhana, Chris Parnin, and Laurie Williams. 2020. Gang of Eight: A Defect Taxonomy for Infrastructure As Code Scripts. In *Proceedings of the 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. to appear, pre-print: https://akondrahman.github.io/papers/icse20_acid.pdf.

[27] Akond Rahman, Effat Farhana, and Laurie Williams. 2020. The 'as Code' Activities: Development Anti-patterns for Infrastructure as Code. *Empirical Softw. Engg.* (2020), 43. https://doi.org/10.1007/s10664-020-09841-8 to appear, pre-print: https://arxiv.org/pdf/2006.00177.pdf.

[28] Akond Rahman, Rezvan Mahdavi-Hezaveh, and Laurie Williams. 2018. A systematic mapping study of infrastructure as code research. *Information and Software Technology* (2018). https://doi.org/10.1016/j.infsof.2018.12.004

[29] Akond Rahman, Chris Parnin, and Laurie Williams. 2019. The seven sins: Security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 164–175.

[30] Akond Rahman, Asif Partho, Patrick Morrison, and Laurie Williams. 2018. What Questions Do Programmers Ask About Configuration As Code?. In *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering* (Gothenburg, Sweden) *(RCoSE '18)*. ACM, New York, NY, USA, 16–22. https://doi.org/10.1145/3194760.3194769

[31] Akond Rahman, Md. Rayhanur Rahman, Chris Parnin, and Laurie Williams. 2020. Security Smells in Ansible and Chef Scripts: A Replication Study. *ACM Trans. Softw. Eng. Methodol.* (2020), 31. To appear. pre-print: https://arxiv.org/pdf/1907.07159.pdf.

[32] Akond Rahman and Laurie Williams. 2018. Characterizing Defective Configuration Scripts Used for Continuous Deployment. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. 34–45. https://doi.org/10.1109/ICST.2018.00014

[33] Akond Ashfaque Ur Rahman, Eric Helms, Laurie Williams, and Chris Parnin. 2015. Synthesizing continuous deployment practices used in software development. In *2015 Agile Conference*. IEEE, 1–10.

[34] Johnny Saldaña. 2015. *The coding manual for qualitative researchers*. Sage.

[35] David Schmitt. 2016. Hitchhiker's guide to testing infrastructure as/and code — don't panic! https://puppet.com/blog/hitchhikers-guide-to-testing-infrastructure-as-and-code/. [Online; accessed 20-Jun-2020].

[36] Julian Schwarz. 2017. Code Smell Detection in Infrastructure as Code. https://www.swc.rwth-aachen.de/thesis/code-smell-detection-infrastructure-code/. [Online; accessed 02-July-2019].

[37] S. Shamim, F. Bhuiyan, and A. Rahman. 2020. In *2020 IEEE Cybersecurity Development (SecDev)*. to appear, pre-print: https://arxiv.org/pdf/2006.15275.pdf.

[38] Tushar Sharma, Marios Fragkoulis, and Diomidis Spinellis. 2016. Does Your Configuration Code Smell?. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 189–200. https://doi.org/10.1145/2901739.2901761

[39] Will Thames and others of Redhat. 2018. Ansible Lint Documentation. https://docs.ansible.com/ansible-lint/. [Online; accessed 20-Jun-2020].

[40] James Turnbull. 2007. *Pulling Strings with Puppet: Automated System Administration Done Right*. Apress.

[41] Akond Ashfaque Ur Rahman and Laurie Williams. 2016. Software Security in DevOps: Synthesizing Practitioners' Perceptions and Practices. In *Proceedings of the International Workshop on Continuous Software Evolution and Delivery* (Austin, Texas) *(CSED '16)*. ACM, New York, NY, USA, 70–76. https://doi.org/10.1145/2896941.2896946

[42] Eduard van der Bent, Jurriaan Hage, Joost Visser, and Georgios Gousios. 2018. How good is your puppet? An empirically defined and validated quality model for puppet. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 164–174. https://doi.org/10.1109/SANER.2018.8330206

[43] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 1–10.